UNITED STATES PATENT APPLICATION

FOR

DOCUMENT VERSION INTEROPERABILITY

INVENTORS:

Chris Ingersoll, a citizen of the United States

ASSIGNED TO:

Commerce One Operations, Inc., a Delaware Corporation

PREPARED BY:

THELEN, REID, & PRIEST, LLP
P.O. BOX 640640
SAN JOSE, CA 95164-0640
TELEPHONE: (408) 292-5800
FAX: (408) 287-8040

Attorney Docket Number: COMM-005

Client Docket Number: COMM-005

SPECIFICATION

TITLE OF INVENTION

DOCUMENT VERSION INTEROPERABILITY

FIELD OF THE INVENTION

[0001] The present invention relates to the versioning of software documents. More specifically, the present invention relates to document version interoperability.

BACKGROUND OF THE INVENTION

[0002] Computer software programs often produce documents. Examples of such programs include word processors, spreadsheets, and graphic creators, among others. Documents normally contain data saved in a format that may or may not be specific to the piece of software used to create it. For example, a word processor may save a document in a format that only that word processor could read, or some word processors may have the ability to save a document in a format that other types of word processors or even other programs can read.

[0003] An e-commerce community comprises a number of entities, normally various businesses or applications within a single business, who exchange business documents with each other. Examples of documents typically exchanged in e-commerce communities include purchase orders, requests for quotes, and sales confirmations, among others. The entities exchanging the documents typically include trading partners, internal applications, and business services.

[0004] Exchange of these business documents normally is accomplished by defining a structure and representing the business logic in documents based on that structure. Often, a markup language, such as Extensible Markup Language (XML) is used. The documents may be wrapped in electronic messages and exchanged over an e-marketplace.

[0005] Each business document exchange may represent a named portion of a business transaction. However, the potential logic represented by the named business document may evolve over time. In the case of XML-based documents, a schema or DTD is updated and available data elements may be added, removed, or changed in new incarnations of the logic. This evolution of business document structure is called versioning.

[0006] Each new structure defines a new version of that business document. However, when dealing in an e-marketplace, it is quite common that one or more members of the trading community does not natively support all versions of all business documents traded in their community. This creates a situation where documents may be sent to entities that do not understand their structure.

[0007] Historically, formats used for the exchange of information such as Electronic Data Exchange (EDI) have solved this problem in one of two ways. The first solution is to define a community version and require that all participating entities (trading partners, internal applications, business services) comply with that community version. Two problems with this approach are potential data loss and synchronized migration.

[0008] A data loss example is two entities that natively support the same higher level document than the community version and translate to the community version before sending and back after receiving. If all the logic in the higher level document is not representable in the lower, this document exchange is not as rich as a pure native version exchange.

[0009] Another problem with this approach is that it requires potentially expensive coordination between entities to synchronize migration to a new document version.

[00010] The second solution is to force the receiver of the document to support all possible sender formats, perhaps by translating to a version he can understand. This solution also requires coordination migration of a community to a new version. If one member migrates to a new versions, all potential receivers from this member must be able to support this new version (perhaps via translation).

[00011] What is needed is a solution that allows for the exchange of documents with minimal data loss while still providing for independent migration by the participants.

BRIEF DESCRIPTION OF THE INVENTION

[00012] Document version interoperability is provided by allowing members of a community to maintain independent migration by permitting the members to continue to run native application software on their respective systems. A community may define a community version by establishing certain rules for documents. When electronically transmitting a document, a member of the community may provide in the transmitted message containing the document his native version of the document, the community version of the document, as well as any or all versions of the document which are closer to the community version of the document than his native version of the document. This may be accomplished by performing document transformations when creating the message. Upon receipt of the documents, the recipient may choose the document version contained in the message that is most easily read by the recipient's native application program and transform it so that it may be opened by the recipients native application program if necessary. Regardless of what rules are established to define the community version, data loss in any document exchange is minimized. Entities that follow these rules can migrate their native support without requiring coordination with other entities. Members do not have to know the native version supported by other members. This ensures privacy for the members and also lessens the need for direct communications between the members.

BRIEF DESCRIPTION OF THE DRAWINGS

[00013] The accompanying drawings, which are incorporated into and constitute a part of this specification, illustrate one or more embodiments of the present invention and, together with the detailed description, serve to explain the principles and implementations of the invention.

[00014] In the drawings:

FIG. 1 is a flow diagram illustrating a method for sending a document to a community member in accordance with a specific embodiment of the present invention.

FIG. 2 is a flow diagram illustrating a method for receiving a document from a community member in accordance with a specific embodiment of the present invention.

FIG. 3 is a block diagram illustrating an apparatus for sending a document to a community member in accordance with a specific embodiment of the present invention.

FIG. 4 is a block diagram illustrating an apparatus for receiving a document from a community member in accordance with a specific embodiment of the present invention.

DETAILED DESCRIPTION

[00015] Embodiments of the present invention are described herein in the context of a system of computers, servers, communication mechanisms, and tags. Those of ordinary skill in the art will realize that the following detailed description of the present invention is illustrative only and is not intended to be in any way limiting. Other embodiments of the present invention will readily suggest themselves to such skilled persons having the benefit of this disclosure. Reference will now be made in detail to implementations of the present invention as illustrated in the accompanying drawings. The same reference indicators will be used throughout the drawings and the following detailed description to refer to the same or like parts.

[00016] In the interest of clarity, not all of the routine features of the implementations described herein are shown and described. It will, of course, be appreciated that in the development of any such actual implementation, numerous implementation-specific decisions must be made in order to achieve the developer's specific goals, such as compliance with application- and business-related constraints, and that these specific goals will vary from one implementation to another and from one developer to another. Moreover, it will be appreciated that such a development effort might be complex and time-consuming, but would nevertheless be a routine undertaking of engineering for those of ordinary skill in the art having the benefit of this disclosure.

[00017] In accordance with the present invention, the components, process steps, and/or data structures may be implemented using various types of operating systems, computing platforms, computer programs, and/or general purpose machines. In addition, those of ordinary skill in the

art will recognize that devices of a less general purpose nature, such as hardwired devices, field programmable gate arrays (FPGAs), application specific integrated circuits (ASICs), or the like, may also be used without departing from the scope and spirit of the inventive concepts disclosed herein.

[00018] Document version interoperability is provided by allowing members of a community to maintain independent migration by permitting the members to continue to run native application software on their respective systems. It allows members to maintain independent migration by allowing them to continue to run native software on their systems. A community may define a community version by establishing certain rules for documents. When electronically transmitting a document, a member of the community may provide in the transmitted message containing the document his native version of the document, the community version of the document, as well as any or all versions of the document which are closer to the community version of the document than his native version of the document. This may be accomplished by performing document transformations when creating the message. Upon receipt of the documents, the recipient may choose the document version contained in the message that is most easily read by the recipient's native application program and transform it so that it may be opened by the recipients native application program if necessary. Regardless of what rules are established to define the community version, data loss in any document exchange is minimized. Entities that follow these rules can migrate their native support without requiring coordination with other entities. Members do not have to know the native version supported by other members. This ensures privacy for the members and also lessens the need for direct communications between the members.

[00019] Another advantage the present invention provides is that the recipients don't necessarily have to know about this scheme. If they support the community version of the document natively, then they may simply ignore any extraneous documents contained within the message. Only recipients who do not natively support the community version of the document need to implement the present invention, and all recipients, no matter which version of the document they support natively, may then exchange information easily.

[00020] The present invention will be discussed in terms of its application to an e-commerce community exchanging business documents. However, one of ordinary skill in the art will recognize that implementations are possible involving data exchange in general and any type of document that needs to be share between two or more parties.

[00021] In the present invention, a community version is defined for the structure of each business document. Typically, this community version will be the most recent version available, but there may be cases where a less recent version is more desirable. Members are capable of transforming between their native version of the application software and the community version as well as all versions that are closer to the community version than their native version. Closeness is a concept that will be discussed in greater detail later in this application.

[00022] When sending a document, members perform transformations between their native version of the document, the community version of the document, and all versions of the document closer to the community version of the document than their native version of the document. All of these transformations are then packed into a message and sent. Upon receipt

of the message, the receiving member may then choose the document version closest to the natively supported version. A transformation may be performed if necessary.

[00023] Closeness may be defined as the amount of data loss that is incurred when transforming between versions of documents. If one version of a document is closer than another, it can be transformed with less data loss.

[00024] FIG. 1 is a flow diagram illustrating a method for sending a document to a community member in accordance with a specific embodiment of the present invention. The document may be saved in a version native to the member executing the method. At 100, it is determined if the native version is equivalent to a community version of the document. If so, at 102, the native version of the document is sent to the community member. If the native version of the document is not equivalent to the community version of the document, at 104 the native version of the document is converted to the community version of the document and to any versions of the document closer to the community version than the native version. Then, at 106, the native version, transformed community version, and any other transformed version of the document are all sent to the community member.

[00025] FIG. 2 is a flow diagram illustrating a method for receiving a document from a community member in accordance with a specific embodiment of the present invention. At 200, a message is received, the message containing one or more documents each having a version. At 202, it is determined if the message has only one document. If it does, then that document must be the community version of the document, and thus at 204 the document may be used,

converting to a native version of the document if necessary. If there was more than one document in the message, then at 206 it is determined which document is closest to the native version of the document. That closest document is then used at 208, again converting it to the native version of the document if necessary.

[00026] The present invention may be implemented using a versioning library interfacing with the community. The versioning library may be included with a software produce sold for portal connections. This implementation will be discussed herein, but other implementations are possible within the scope of the disclosure.

[00027] A library of schema representation for business documents may be utilized. This library keeps track of all the possible schemas for the business documents used in the community. For example, there may be three different schema for business documents, known as versions 2.0, 2.2, and 3.0. These three schemas may be stored in this library.

[00028] A special type of enveloping scheme (known herein as MarketSite Message Layer, or MML) may be used to allow a primary document to be accompanied by any number of attachments when it is sent. The versioning library may put the community version as the primary document, and the alternate versions as attachments using a consistent attachment naming scheme that encompasses the document type and version of the attachment.

[00029] A message may hold one, and only one document, whereas other documents are added as attachments. Attachments may be XML documents as well as any other type of format.

Messages may also contain a property list with a key, value pairs, a context document, and a catalog document used to resolve references to attachments.

[00030] Messages may have properties, which may be used for routing and/or bookkeeping. This design differentiates between managed properties and user-provided properties. Managed properties are written once and then from that point on are read-only. User provided properties have no such limitations. In a specific embodiment of the present invention, properties are richer than the default Java properties class as they can have an associated parameter list.

[00031] Different properties may be set by different places in the system and at different times. Table 1 below illustrates a list of potential properties, where they are set in the system, when they are set, and why they are set.

Property	Who?	When?	Why?
x-Document-Type	Message	Message creation time	To enable fast routing on the server.
x-Message-Id	Message	Message creation time	To track messages.
x-Correlation-Id	Service	Service has reply document ready, want to publish back	To track messages, and resulting messages.
x-Request-Mode	Transmitter	Set when the message is passed in.	To let the sender specify processing hints related to the request. Read more below.
x-Date-Received	Server Agent	When the Message reaches the server.	To keep some bookkeeping regarding dates. System and legal reasons.
x-Date-Sent	Transmitter closest to the wire. E.g. an InternalPublisher doesn't set this property.	Just before the Message is sent over the wire	To keep some bookkeeping regarding dates. System and legal reasons.
x-Receiver-Id	Transmitter returned from first lookup.	Lookup required receiver info, stored in resulting Transmitter instance. Set when the message is passed in.	To make sure the Message reaches the right destination. May it be a hosted, or integrated service. Used for routing on the server.
x-Sender-Id	Transmitter returned from first lookup.	Set when the message is passed in.	To make sure the recipient has enough knowledge to lookup info it needs, e.g. preferred callback address.

Table 1

[00032] The message property x-Request-Mode is used to hold processing hints. A hint is designed to override any default values the receiver has stored or looked-up. Hints may be ignored due to transport, or to server policies.

[00033] Constants for the keys of the managed properties may then be stored in a general class. An application developer may add properties for its own processing. However, policies to guarantee uniqueness may have to be introduced if this is the case. This may be accomplished by using the general class created for the keys of the managed property as a instance called by the class defining message properties in its constructor. It then may use the value of a string that describes the managed keys to decide which properties are managed and which are user defined.

[00034] The catalog document described earlier is maintained in the message to be the first data used when resolving references within the document. For example, a document could be referring to an attachment in the same message. The catalog will then help to resolve that relationship.

[00035] The context document described earlier may be carried within the message to keep the current context available. The context may have relevance to security, document exchange protocols, and transactions.

[00036] There may also be two different levels of support for attachments. The first may be to stored attachments in the message itself. The second may be to have a Universal Resource Identifier (URI) be bound to an element in the document. This URI may be used to bind the attachment in the message. This second level of support may be called "Named and Bound attachments", whereas the first level of support may be simply called "attachments". An iterator on the message may be provided when named and bound attachments are not used.

[00037] On the client side, a programmer is concerned with creating the message and sending it to the business partner for processing by a business service. When a named and bound attachment is used, the developer needs to the set the reference attribute on the element. In doing so, a URI is used, the same URI that will later be used when adding the attachment to the message. Alternatively, link classes may be implemented to create an even stronger binding.

[00038] When the message is received by the server side, the element is the same as was created at the client side, except that a few more properties may have been added along the way.

[00039] The versioning library also has two settings for the versions of each document, internal and external version. The internal version defines the native version used by an endpoint. The external version is the community version. The versioning library is invoked when the messages are sent or received in order to modify the messages in accordance with the rules.

[00040] A separate transformation registry may be used to store the transformation logic between various versions of document types. In a specific embodiment of the present invention, a table is used for the transformation registry. Entries in this table may define the linkages between documents in separate version and identify the Java class or Extensible Syntax Language Transformation (XSLT) file that performs the transformation.

[00041] Closeness may be defined in a number of different ways. In a specific embodiment of the present invention, the versioning library assumes that there is no data loss from a lower

version to a higher version of a document, but there is data loss when converting from a higher version to a lower of a document. Thus, when determining the closest version of a document to a given document version, the system may first look to available higher version numbers of the document, and then take the mathematically closest higher version to the given document version. Only if there are no available higher version numbers will the system look to lower numbers, taking the mathematically closest lower version to the given version. This assumes a decimal or other mathematical numbering scheme, but one of ordinary skill in the art will recognize that embodiments are possible with other types of versioning schemes, such as using letters, codes, or labels.

[00042] FIG. 3 is a block diagram illustrating an apparatus for sending a document to a community member in accordance with a specific embodiment of the present invention. The document may be saved in a version native to the member executing the method. A native version/community version equivalence determiner 300 determines if the native version of the document is equivalent to a community version of the document. If so, a native version document sender 302 coupled to the native version/community version equivalence determiner 300 sends the native version of the document to the community member. If the native version of the document is not equivalent to the community version of the document, a native version document converter 304 coupled to the native version/community version equivalence determiner 300 converts the native version to the community version and to any versions closer to the community version than the native version. Then, a native/community/other converted document sender 306 coupled to the native version document converter 304 sends the native version, transformed community version, and any other transformed version of the document to

the community member. The native/community/other converted document sender 306 may include a community version document message encapsulator 308, which encapsulates the community version document in a message, and a native version/other converted document attachment saver 310 coupled to the community version document message encapsulator 308, which saves the native version document and any other converted document as an attachment to the message. A transformation registry 312 may contain information as to how to transform documents between versions.

[00043] FIG. 4 is a block diagram illustrating an apparatus for receiving a document from a community member in accordance with a specific embodiment of the present invention. A message receiver 400 receives a message, the message containing one or more documents each having a version. A one document message determiner 402 coupled to the message receiver determines if the message has only one document. If it does, then that document must be the community version, and thus the document may be used, converting to a native version of the document if necessary using a community version to native version converter 404 coupled to the message receiver 400 and the one document message determiner. If there was more than one document in the message, then a closest document determiner 406 coupled to the community version to native version converter 404 determines which document is closest to the native version. That closest document is then used, converting it to the native version using a closest document to native version transformer 408 coupled to the message receiver 400 and the closest document determiner 406 if necessary. A transformation registry 410 may contain information as to how to transform documents between versions.

[00044] While embodiments and applications of this invention have been shown and described, it would be apparent to those skilled in the art having the benefit of this disclosure that many more modifications than mentioned above are possible without departing from the inventive concepts herein. The invention, therefore, is not to be restricted except in the spirit of the appended claims.